RVCAT: Simulador web interactiu per a l'estudi del rendiment de processadors

Aleix Jordà Banús

9 de febrer de 2026

Resum– Aquest projecte presenta el desenvolupament i la millora de RVCAT-WEB, un simulador de rendiment web dissenyat per facilitar l'aprenentatge de l'arquitectura de processadors. El treball se centra en la implementació d'un sistema de tutorials interactius, un editor visual per a la creació de programes i tutorials, una eina de comparació de mètriques i millores en l'apartat de *cache*.
L'aplicació utilitza tecnologies web com Vue.js i Pyodide, funcionant completament al navegador sense servidor (*client-side*). S'ha implementat un sistema d'emmagatzematge local per permetre la persistència de dades entre sessions i unificar la gestió de programes i configuracions.

Paraules clau– arquitectura de computadors, simulador, aplicació web, Pyodide, Vue.js, *cache*, tutorial interactiu.

Enllaç: `https://hpca4se-uab.github.io/rvcat-web-ui/`

Abstract– This project presents the development and improvement of RVCAT-WEB, a web-based performance simulator designed for learning processor architecture. The work focuses on the implementation of an interactive tutorial system, a visual editor for creating programs and tutorials, a metrics comparison tool, and improvements to the cache section.
The application uses Vue.js and Pyodide to run entirely in the browser without a server. A local storage system has been implemented to persist data between sessions and unify the management of programs and configurations.
Keywords– computer architecture, simulator, web application, Pyodide, Vue.js, cache, interactive tutorial.

Link: `https://hpca4se-uab.github.io/rvcat-web-ui/`

1 Introducció - Context del treball

AQUEST treball pren com a punt de partida un simulador de rendiment d'arquitectura de processadors que permet l'execució i l'anàlisi de programes en codi de baix nivell. La plataforma actual permet als usuaris carregar programes escrits en un llenguatge assemblador genèric i editar la configuració de l'arquitectura del processador.

A més, ofereix la possibilitat de modificar paràmetres fonamentals com la mida del *Reorder Buffer* (ROB), una estructura que permet l'execució fora d'ordre d'instruccions, o el nombre d'iteracions de la simulació.

Per a l'anàlisi de rendiment, el simulador ja compta amb eines de visualització. D'una banda, permet observar el *timeline* d'execució de les instruccions, facilitant la comprensió del seu flux i paral·lelisme. D'altra banda, incorpora una funcionalitat per analitzar el graf de dependències entre instruccions, identificant camins crítics i colls d'ampolla.

Finalment, calcula mètriques essencials com l'IPC (*Instructions Per Cycle*), que mesura l'eficiència del processador, i altres estadístiques derivades de la simulació.

Tot i això, hi ha diversos aspectes en els quals el simulador podria ser més complet. L'estructura actual presenta múltiples components tècnics (configuració d'arquitectura, editor de processadors, anàlisi de dependències, visualització de *timeline*) que poden dificultar l'ús per a usuaris novells. La manca d'una guia porta els estudiants a explorar l'eina sense ordre, perdent l'oportunitat d'aprendre conceptes fonamentals de manera estructurada. Per aquest motiu, un sistema de tutorials interactius que guiï l'usuari en

• E-mail de contacte: Aleix.Jorda@autonoma.cat
• Menció: Enginyeria de Computadors
• Treball tutoritzat per: Juan Carlos Moure Lopez i Albert Jiménez Blanco
• Curs 2025/26

l'ús de les diferents funcions seria una millora rellevant. A més, aquests tutorials podrien ser utilitzats pel professorat per crear activitats o lliçons complementàries a les classes. Un altre aspecte que es va considerar durant el desenvolupament és la visualització de l'estat de la memòria *cache* i la distribució de dades a memòria, informació que seria valuosa per entendre el comportament del processador i els colls d'ampolla associats a l'accés a dades.

També es considera rellevant incorporar una funcionalitat que permeti comparar les mètriques de rendiment de diverses execucions per analitzar l'efecte de diferents configuracions, així com disposar d'un sistema d'emmagatzematge local que permeti guardar i recuperar resultats de simulacions entre sessions. En aquesta línia, s'ha reorganitzat el flux de dades del sistema: els programes i configuracions de processadors, que anteriorment residien al mòdul Python, ara es gestionen directament des de la capa web, fet que en simplifica el manteniment i l'extensibilitat.

L'eina s'ha provat durant un període prolongat tant per alumnes com per professors, cosa que ha permès identificar quines parts resulten més útils i quines funcionalitats addicionals serien més beneficioses per a l'aprenentatge.

1.1 Objectius

Aquest treball té com a objectiu desenvolupar noves funcionalitats per millorar la usabilitat i el valor educatiu del simulador (accessible a `hpca4se-uab.github.io/rvcat-web-ui/`), fent-lo més complet i adequat per a l'aprenentatge d'arquitectura de processadors.
Els objectius principals del projecte són:

- Simplificar l'arquitectura movent la gestió de programes i processadors del mòdul Python a la capa web.
- Implementar un sistema d'emmagatzematge en el navegador per guardar programes i configuracions per garantir la persistència de les dades en altres funcionalitats.
- Implementar un sistema de tutorials interactius que guiï l'usuari i permeti al professorat crear lliçons. Permetre la inclusió d'imatges i preguntes tipus test, així com un editor per crear tutorials.
- Integrar un editor de programes per poder editar i crear programes directament des de la web.
- Visualitzar els resultats de les execucions en una pestanya de comparació de mètriques. Aquesta informació es podrà salvar per a futures sessions i exportar en format *.csv*.
- Afegir una pestanya per a l'estat de la memòria *cache* i visualització de memòria.

2 METODOLOGIA

S'ha adoptat una metodologia *Agile* per al desenvolupament d'aquest projecte, centrada en la iteració ràpida. Aquesta aproximació resulta especialment adequada per a projectes on els requisits poden canviar durant el curs del projecte. També permet disposar sempre d'una versió funcional o molt propera a ser-ho, cosa que permet utilitzar l'eina al mateix temps que es treballa en noves funcionalitats.

El desenvolupament del projecte s'ha estructurat en diverses fases, cadascuna centrada en la implementació d'una funcionalitat específica del simulador. A continuació es detallen els aspectes tècnics més rellevants de cada component desenvolupat.

Respecte a la planificació inicial, s'han dut a terme algunes modificacions. S'ha implementat una característica al tutorial interactiu que no s'havia plantejat inicialment. Concretament, s'ha afegit l'opció d'incorporar preguntes de tipus test en el tutorial. A més, la revisió del codi existent va permetre detectar oportunitats de millora, com la reorganització del flux de dades descrita a continuació.

3 ARQUITECTURA DEL SISTEMA

En aquesta secció es descriu l'estructura tècnica de RVCAT-WEB, explicant els components clau i com interactuen per executar el simulador directament al navegador. Primer s'aborden les tecnologies web que suporten la interfície, i després s'explica com Pyodide permet carregar i executar codi Python en un entorn de client.

3.1 Tecnologies Web

L'aplicació utilitza HTML, JavaScript i CSS (estils) com a tecnologies base. Sobre aquestes, s'empra Vue.js 3 com a *framework* de *frontend*, que facilita el desenvolupament mitjançant una metodologia basada en components. Vue.js organitza el codi seguint el patró MVVM (Model-View-ViewModel): les plantilles HTML mostren la interfície (View), les funcions reactives gestionen l'estat i les interaccions (ViewModel), i les dades del simulador representen el model (Model). Aquesta separació facilita el manteniment del codi.

3.2 Execució de Python al Navegador

RVCAT-WEB utilitza Pyodide per executar el simulador RVCAT (escrit en Python) directament al navegador, sense necessitat de servidor. Pyodide és una distribució de Python que compila el codi Python a WebAssembly, una tecnologia que permet executar codi de llenguatges de programació tradicionals (com Python, C++ o Rust) dins del navegador web. El procés comença amb un fitxer `.whl` (*wheel*), un format estàndard per distribuir paquets Python que conté el codi i les seves dependències. Pyodide interpreta aquest fitxer, el carrega a la seva màquina virtual Python, i l'executa mitjançant WebAssembly.

Amb aquesta arquitectura, tota l'aplicació funciona al navegador. La pàgina es desplega automàticament a GitHub Pages mitjançant GitHub Actions, garantint l'accés a la versió de producció sense gestionar infraestructura addicional.

3.3 Versió Original i Simplificació

En la versió original (Figura 1), les definicions de programes i configuracions de processadors residien dins del mòdul Python (RVCAT). Això implicava que qualsevol modificació requeria reconstruir el paquet `.whl` i tornar-lo a desplegar.

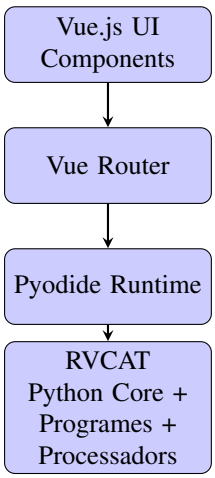

Fig. 1: Arquitectura original: programes i processadors dins del mòdul Python

Per millorar el manteniment, s'ha traslladat la gestió de programes i processadors a la capa web: ara es defineixen en fitxers JSON accessibles directament des del *frontend* (Figura 2). Les dades es guarden al *LocalStorage* del navegador per recuperar-les en sessions posteriors.

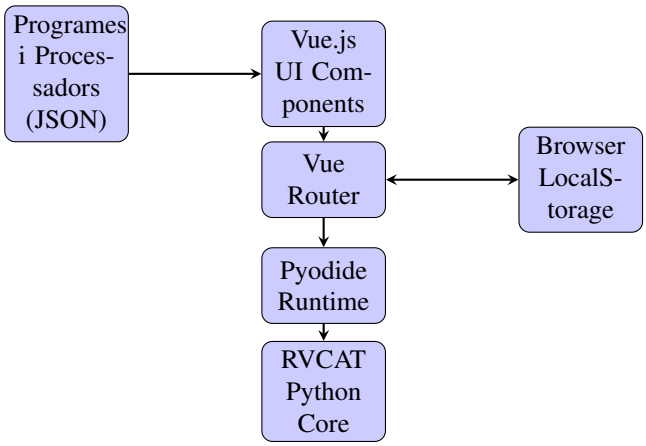

Fig. 2: Arquitectura actual: programes i processadors separats del nucli Python

Aquest canvi ofereix diversos avantatges: afegir o modificar programes es redueix a editar un fitxer JSON sense modificar el codi Python, i es desacobla el nucli de simulació de les dades de configuració, facilitant que diferents usuaris mantinguin els seus propis conjunts de programes.

4 Desenvolupament

En aquesta secció es descriuen les funcionalitats implementades durant el projecte. S'ha prioritzat el sistema d'emmagatzematge local com a base transversal, seguit dels tutorials interactius, l'editor de programes i la comparació de mètriques.

4.1 Sistema d'Emmagatzematge Local

Com a primera funcionalitat transversal, s'ha implementat un sistema de persistència que utilitza *LocalStorage* del navegador. Aquest sistema serveix de base per a la resta de funcionalitats desenvolupades: emmagatzematge de resultats de rendiment globals, progrés dels tutorials, esborranys dels editors de tutorials i programes, i configuracions de processadors. Les subseccions següents descriuen com cada funcionalitat específica s'integra amb aquest mecanisme central.

4.1.1 Funcionament del LocalStorage

LocalStorage és una API del navegador web que permet guardar dades en forma de parelles clau-valor de manera persistent al dispositiu de l'usuari. A diferència de la memòria cau del navegador, que s'utilitza per emmagatzemar recursos web temporalment i que pot ser esborrada automàticament, *LocalStorage* manté les dades fins que siguin eliminades explícitament per l'aplicació o l'usuari.

Pel que fa a la capacitat d'emmagatzematge, el límit és d'aproximadament 5-10 MB per domini [8]. Aquest límit és, en principi, suficient per a RVCAT-WEB, ja que la major part de les dades són text que es guarda en JSON. També hi ha l'opció de carregar imatges, que seria l'únic cas en què es podria arribar al límit.

4.1.2 Estratègia de Serialització i Recuperació

Les dades es serialitzen en format JSON abans de guardar-les, ja que *LocalStorage* només accepta cadenes de text. Això permet emmagatzemar estructures complexes com simulacions completes, esborranys de tutorials o programes personalitzats. El sistema organitza les dades amb claus específiques per a cada tipus d'informació. El Llistat 1 mostra un exemple d'implementació per a diferents funcionalitats.

```
// Guardar execucio
export function saveExecution(execution) {
  const executions = getStoredExecutions();
  executions.push({
    ...execution,
    id: generateExecutionId(),
    timestamp: new Date().toISOString()
  });
  localStorage.setItem('rvcat_executions',
    JSON.stringify(executions));
}

// Guardar esborrany de tutorial
export function saveTutorialDraft(tutorial) {
  localStorage.setItem('rvcat_tutorial_draft',
     JSON.stringify(tutorial));
}

// Guardar programa en edicio
export function saveProgramDraft(program) {
  localStorage.setItem('rvcat_program_draft',
    JSON.stringify(program));
}
```

Listing 1: Exemple d'ús del LocalStorage per diferents funcionalitats

La funció `saveExecution` il·lustra el patró general utilitzat: primer es recuperen els resultats existents amb `getStoredExecutions()`, després s'afegeix la nova execució amb un identificador únic i una marca temporal, i finalment es serialitza tot el conjunt amb `JSON.stringify()` abans de guardar-lo. Aquest patró garanteix que les dades es mantenen consistents entre sessions. Per recuperar les dades, el procés és l'invers: es llegeix la cadena de text amb `localStorage.getItem()`, es

deserialitza amb `JSON.parse()` i s'obté l'objecte JavaScript original.

A més de l'emmagatzematge local, el sistema permet exportar i importar dades en format JSON, facilitant compartir resultats entre diferents usuaris o dispositius. Per importar, l'usuari selecciona un fitxer JSON del seu ordinador; per exportar, tria un directori on desar el fitxer generat. Aquesta funcionalitat complementa el *LocalStorage* quan es vol fer còpia de seguretat, treballar en diferents ordinadors o compartir configuracions específiques.

4.2 Sistema de Tutorials Interactius

El sistema de tutorials és una de les principals funcionalitats del projecte. Està dissenyat per ajudar l'estudiant a entendre l'arquitectura de computadors mitjançant lliçons estructurades. Aquestes combinen dos mecanismes: passos guiats sobre elements de la interfície, amb comprovacions que verifiquen que l'usuari hagi executat l'acció requerida, i preguntes de tipus test integrades al flux del tutorial per consolidar coneixements abans de continuar.

Els tutorials són accessibles des d'un menú desplegable que apareix en prémer el botó "`?`" situat a la cantonada inferior dreta de la interfície (Figura 3). L'usuari pot seleccionar qualsevol tutorial disponible i seguir-lo pas a pas.

Un aspecte important del sistema és que el progrés dels tutorials es manté entre sessions. Si l'usuari abandona un tutorial a mig camí, en tornar a la secció de tutorials se li ofereixen dues opcions: **Resume** per continuar des del pas on el va deixar, o **Stop**, per descartar el progrés guardat i començar el tutorial des del principi o fer-ne un altre. Aquesta funcionalitat permet als estudiants interrompre els tutorials i reprendre'ls en un altre moment sense perdre l'avanç realitzat, tal com es pot observar al desplegable de la Figura 3.

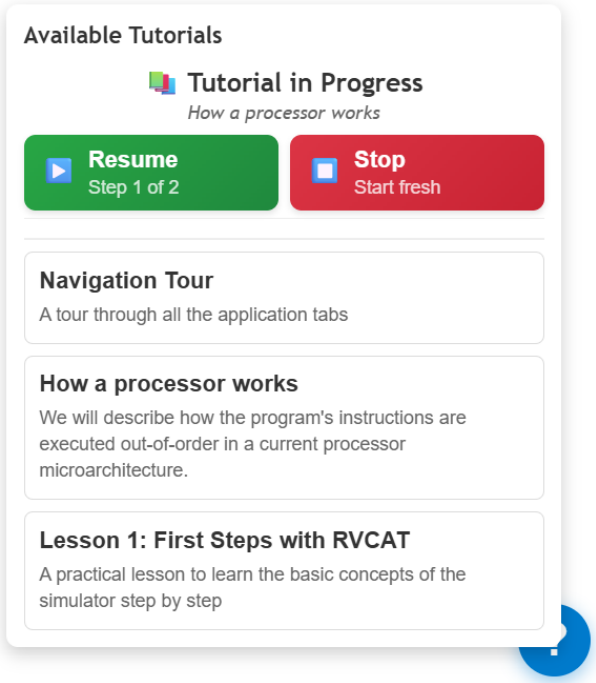

Fig. 3: Menú desplegable de tutorials mostrant la llista disponible i les opcions Resume/Stop

El flux general d'un tutorial segueix l'esquema mostrat a la Figura 4. Cada pas o pregunta es mostra a l'usuari, que ha de completar l'acció requerida o respondre correctament. El sistema valida la resposta: si és correcta, avança al següent pas; si és incorrecta, permet reintentar sense penalització. Aquest cicle continua fins a finalitzar tots els passos del tutorial. Aquest enfocament iteratiu garanteix que l'usuari no pot saltar passos sense haver comprès el contingut, millorant l'efectivitat de l'aprenentatge.

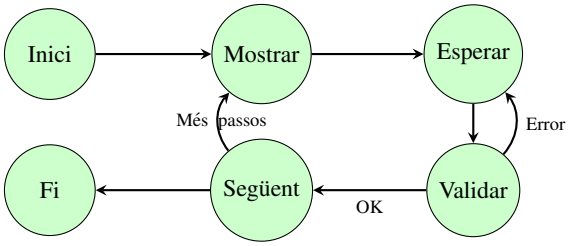

Fig. 4: Flux de validació en tutorials: després de mostrar cada pas, el sistema espera l'acció de l'usuari, valida si és correcta, i només avança si la validació és positiva. En cas d'error, torna a l'estat d'espera permetent reintentar.

4.2.1 Estructura de Dades

Cada tutorial es defineix mitjançant un fitxer JSON que especifica la seqüència de passos, les validacions necessàries i el contingut educatiu. Cada element de la llista `steps` és un objecte amb tipus definit que pot representar un *pas* (`type: step`) o una *pregunta* (`type: question`).

Per als **passos** (`type: step`), l'estructura inclou els següents camps principals:

- `title`: Títol del pas mostrat a l'usuari.
- `description`: Descripció detallada de l'acció a realitzar.
- `selector`: Selector CSS de l'element de la interfície relacionat. Per exemple, "`.header-title`" per seleccionar un element amb la classe `header-title`.
- `position`: Posició del *tooltip* (top, bottom, left, right).
- `validation`: Objecte amb les condicions per avançar al següent pas (opcional).
- `stepImage`: Imatge codificada en base64 per il·lustrar el pas (opcional).
- `action`: Acció a executar, com ara canviar de vista (opcional).

Per a les **preguntes de tipus test** (`type: question`), no es fa servir `selector` ni cap validació d'acció. En comptes d'això, el tutorial mostra una finestra modal centrada i el progrés queda bloquejat fins que l'usuari respongui correctament. Els camps principals són:

- `title`: Títol de la pregunta
- `questionText`: Enunciat (pot incloure HTML per donar format al text)

- `answerMode`: mode de resposta (`single` o `multiple`).
- `answers`: llista d'opcions (fins a 6) amb `text`, `isCorrect` i `explanation`.
- `questionImage`: Imatge codificada en base64 per il·lustrar la pregunta (opcional).

Tant els passos com les preguntes admeten imatges codificades en format base64. Això permet incloure captures de pantalla, diagrames o esquemes explicatius directament al fitxer JSON, sense dependre de recursos externs. Aquesta funcionalitat és especialment útil per mostrar exemples visuals de la microarquitectura o indicar zones concretes de la interfície.

El Llistat 2 mostra un exemple complet d'un pas amb imatge i acció de canvi de vista, mentre que el Llistat 3 il·lustra una pregunta tipus test amb imatge i format HTML en l'enunciat.

```
{
  "type": "step",
  "title": "Microarchitecture of the Execution
     Engine",
  "description": "Look at the main components
    of the microarchitecture.",
  "selector": ".header-title",
  "position": "bottom",
  "stepImage": "
    iVBORw0KGgoAAAANSUhEU...",
  "action": "switchTo:processorEditorComponent
    "
}
```

Listing 2: Exemple d'estructura d'un pas de tutorial amb imatge i acció

```
{
  "type": "question",
  "title": "Waiting Buffer",
  "questionText": "Why do instructions wait in
     the <strong>Waiting Buffer</strong>?",
  "answerMode": "single",
  "answers": [
    {
      "text": "Due to data dependencies or
    execution port unavailability",
      "isCorrect": true,
      "explanation": "Great! Instructions wait
     when operands are not ready."
    },
    {
      "text": "It is a random situation",
      "isCorrect": false,
      "explanation": "No, there are specific
    reasons for waiting."
    }
  ],
  "questionImage": "
    iVBORw0KGgoAAAANSUhEU..."
}
```

Listing 3: Exemple de pregunta tipus test amb imatge i format HTML

4.2.2 Motor de Validació

El motor de validació implementa un sistema modular capaç de verificar diferents tipus d'accions de l'usuari. Cada tipus de validació comprova una condició específica:

- `program_selected`: Verifica que s'ha seleccionat un programa concret.
- `architecture_selected`: Comprova que la configuració d'arquitectura sigui la mateixa que l'especificada.
- `input_value`: Valida que un camp d'entrada té el valor esperat.
- `button_clicked`: Detecta accions com executar la simulació o prémer un botó.

El sistema es basa en un observador d'esdeveniments de Vue que detecta canvis en els components i avalua si es compleixen les condicions de validació del pas actual.

En el cas de les **preguntes tipus test**, el tutorial imposa una condició clara: l'usuari no pot continuar fins que la resposta sigui correcta. Aquest enfocament permet utilitzar les preguntes com a punt de control per consolidar conceptes abans d'avançar. Per reforçar l'aprenentatge, el sistema mostra un missatge sota cada opció seleccionada i permet reintentar fins a aconseguir la resposta perfecta, sense revelar automàticament quines opcions són correctes.

L'ordre de les opcions de resposta es pot alterar mitjançant una permutació aleatòria amb `Math.random()`. En la implementació actual, aquesta barreja s'aplica quan es torna a iniciar el qüestionari (per exemple, en començar de nou tot el tutorial), però no quan l'usuari s'equivoca i prem *Try Again*.

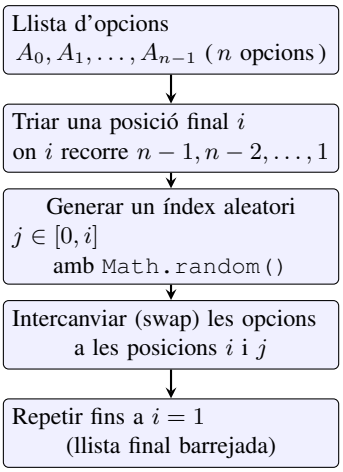

Fig. 5: Esquema de la barreja d'opcions de resposta amb intercanvis successius

Aquesta combinació de guia i avaluació formativa és coherent amb les recomanacions habituals en entorns d'aprenentatge digital, on el *feedback* immediat i la pràctica distribuïda milloren la retenció i la comprensió dels continguts [6].

4.2.3 Editor de Tutorials

Per facilitar la creació de contingut educatiu, s'ha desenvolupat un editor visual que permet al professorat dissenyar tutorials sense necessitat de modificar directament els fitxers JSON. L'editor proporciona una interfície intuïtiva amb formularis per especificar els diferents camps de cada element del tutorial.

L'editor permet afegir i eliminar elements. Cada element pot ser de tipus `step` (pas guiat) o `question` (pregunta tipus test). Els camps obligatoris varien segons el tipus:

- **Per a *steps***: títol, descripció i l'element que cal ressaltar (selector CSS). L'element a ressaltar es pot seleccionar d'una llista predefinida o escriure manualment.
- **Per a *questions***: títol, text de la pregunta, mode de resposta (*single*/*multiple*) i llista de respostes amb indicació de validesa.

Les imatges es poden afegir tant a *steps* com a *questions* mitjançant un explorador de fitxers. Un cop seleccionada, la imatge es converteix automàticament a format base64 i es mostra en previsualització. Hi ha l'opció d'ampliar la mida de visualització de la imatge en ambdós tipus d'elements.

L'editor inclou validadors que comproven que tots els camps obligatoris estan emplenats i que les preguntes tenen almenys dues opcions amb una marcada com a correcta.

4.2.4 Integració amb *LocalStorage*

El sistema de tutorials implementa dues funcionalitats de persistència complementàries mitjançant *LocalStorage*:

Progrés dels tutorials: Quan un estudiant abandona un tutorial a mig camí, el sistema guarda automàticament el pas actual i l'estat de validació. En tornar a la secció de tutorials, l'usuari pot triar entre reprendre des d'on es va quedar o començar de nou. Aquesta persistència millora l'experiència d'aprenentatge permetent sessions d'estudi flexibles.

Esborrany de l'editor: L'esborrany dels tutorials en creació es guarda automàticament després de cada modificació, evitant la pèrdua de feina en cas de tancament accidental del navegador. Quan el professorat torna a obrir l'editor, pot continuar des d'on ho havia deixat. Un cop validat completament, el tutorial es pot exportar com a fitxer JSON compatible amb el sistema.

Les captures de l'editor es mostren a l'Annex (Figures 8 i 9).

4.3 Editor de Programes

L'editor de programes permet crear i modificar seqüències d'instruccions directament des de la interfície web, sense necessitat d'editar fitxers externs. Aquesta funcionalitat resulta especialment útil per a l'aprenentatge, ja que l'usuari pot experimentar amb diferents configuracions de codi i observar immediatament l'impacte sobre el rendiment. La Figura 6 mostra la interfície de la pestanya d'edició de programes.

4.3.1 Estructura d'Instruccions

Cada instrucció es defineix mitjançant un conjunt de camps: un text descriptiu en llenguatge d'alt nivell, un tipus principal amb subtipus, un operand de destinació, fins a tres operands font i un camp per a constants. El sistema organitza els tipus en categories principals (`INT`, `BRANCH`, `MEM`, `FLOAT`, `VFLOAT`) amb subtipus específics per a cada operació. El Llistat 4 mostra l'estructura JSON d'una instrucció que té el tipus `MEM.LOAD` i el subtipus `SP` (*simple precision*).

```json
{
  "type": "MEM.LOAD.SP",
  "text": "tmp = V[i]",
  "destin": "tmp",
  "source1": "i",
  "source2": "",
  "source3": "",
  "constant": "V"
}
```

Listing 4: Estructura JSON d'una instrucció

4.3.2 Interfície i Funcionalitats

La interfície presenta una taula editable on cada fila correspon a una instrucció. Els camps de tipus i subtipus utilitzen menús desplegables que filtren les opcions disponibles de manera jeràrquica: en seleccionar un tipus principal, el segon selector mostra només els subtipus compatibles. Aquesta aproximació redueix errors de configuració i accelera l'edició.

L'editor incorpora botons per afegir, eliminar i reordenar instruccions. Per gestionar fitxers, l'editor ofereix funcions de càrrega i descàrrega. La càrrega permet importar programes des de fitxers JSON externs, mentre que la descàrrega genera el fitxer i permet a l'usuari desar-lo al seu dispositiu.

4.3.3 Integració amb LocalStorage

L'editor implementa un sistema de persistència automàtica que guarda l'estat actual al *LocalStorage* del navegador després de cada modificació. Això garanteix que el treball en curs es preservi fins i tot si es tanca accidentalment el navegador. El programa en edició es recupera automàticament en tornar a obrir l'editor.

4.3.4 Validació i Confirmació

Abans de carregar un programa nou o esborrar l'actual, el sistema comprova si hi ha canvis sense desar. En cas afirmatiu, es mostra un diàleg de confirmació que adverteix l'usuari i li permet cancel·lar l'operació o continuar. Aquesta precaució evita pèrdues accidentals de treball.

Addicionalment, en desar un programa amb un nom ja existent a la llista, el sistema impedeix la sobreescriptura i sol·licita un nom alternatiu. Això garanteix la integritat dels programes preexistents i facilita la gestió de múltiples versions.

4.4 Comparació de Mètriques

La funcionalitat de comparació permet analitzar l'impacte de diferents configuracions sobre el rendiment. Es mostren taules comparatives de mètriques clau com IPC, cicles d'execució i instruccions processades, i es poden seleccionar diverses execucions guardades per veure-les en paral·lel. La Figura 7 mostra la interfície del comparador de mètriques de rendiment.

4.4.1 Implementació Tècnica

El component de comparació carrega les execucions des del *LocalStorage* i les presenta en una taula interactiva. Les dades es poden ordenar per qualsevol columna (processador, programa, ROB, iteracions, IPC, etc.) i filtrar per nom

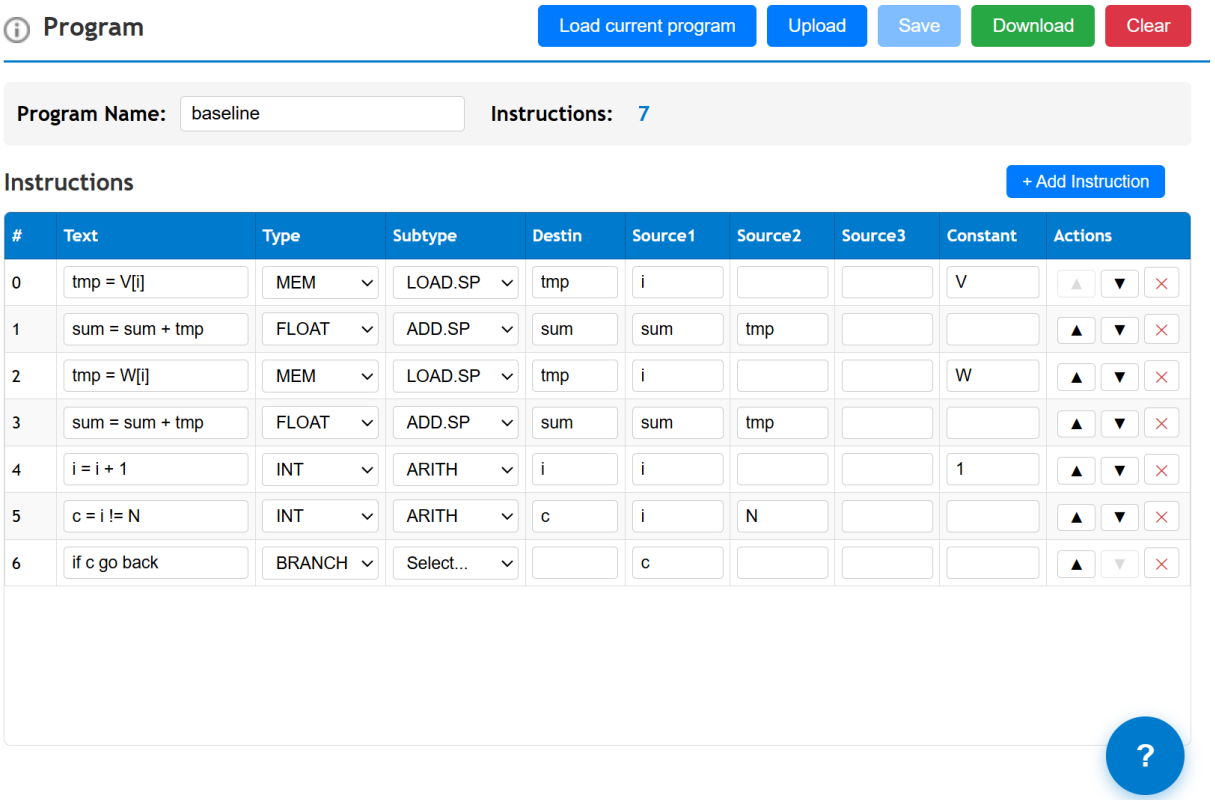

Fig. 6: Interfície de l'editor de programes

o característiques. A més, permet reanomenar i eliminar execucions guardades, descarregar-les en format JSON per enviar-les i carregar-les en altres dispositius, i exportar-les en format CSV per analitzar les mètriques en eines externes com Excel.

4.4.2 Integració amb LocalStorage

Cada vegada que es completa una simulació, els resultats es guarden automàticament al *LocalStorage* amb un identificador únic i marca temporal. L'usuari pot comparar resultats anteriors instantàniament sense haver de tornar a executar les simulacions. Les execucions guardades es mantenen entre sessions del navegador.

5 Línies de millora

Tot i les funcionalitats implementades, hi ha diversos aspectes en què el sistema podria evolucionar per oferir una experiència més completa. Cal destacar que, dels objectius inicials, no s'ha pogut implementar la pestanya de visualització de la memòria *cache*.

5.1 Integració de la memòria cache al mòdul RVCAT

Actualment, la interfície web ja disposa de components preparats per visualitzar i configurar la memòria *cache*, però el mòdul RVCAT no implementa del tot aquesta lògica. Caldria modificar el nucli Python per modelar correctament els accessos a *cache*, incloent-hi la identificació de la mida i posició dels *arrays* a la memòria principal. L'editor de programes ja permet definir instruccions de tipus `MEM.LOAD` i `MEM.STORE` amb constants que representen *arrays*, però la simulació encara no processa aquesta informació per calcular *hits* i *misses*. Aquesta extensió permetria als estudiants entendre millor com la jerarquia de memòria afecta el rendiment.

5.2 Altres millores potencials

A continuació es presenten algunes extensions que podrien enriquir el simulador:

- **Validació de sintaxi a l'editor de programes**: Implementar una comprovació en temps real que detecti errors de sintaxi mentre l'usuari edita les instruccions. El *feedback* immediat reduiria errors tipogràfics i evitaria problemes durant la simulació.
- **Revalidació en reprendre tutorials**: Actualment, quan es reprèn un tutorial guardat, el sistema no verifica si les condicions de validació dels passos anteriors encara es compleixen. Això pot generar incoherències si l'estat de l'aplicació ha canviat entre sessions. Una millora seria implementar una revalidació automàtica en reprendre un tutorial.
- **Gestió del límit del *LocalStorage***: Tot i que el límit típic de 5-10 MB és suficient per a la majoria de casos, seria convenient implementar un sistema de control que monitoritzi l'espai utilitzat.
- **Visualització iteració per iteració**: Permetre observar l'evolució del programa pas a pas, mostrant l'estat de la *cache* i les mètriques en cada iteració. Això ajudaria a comprendre com canvia el comportament del sistema al llarg de l'execució.

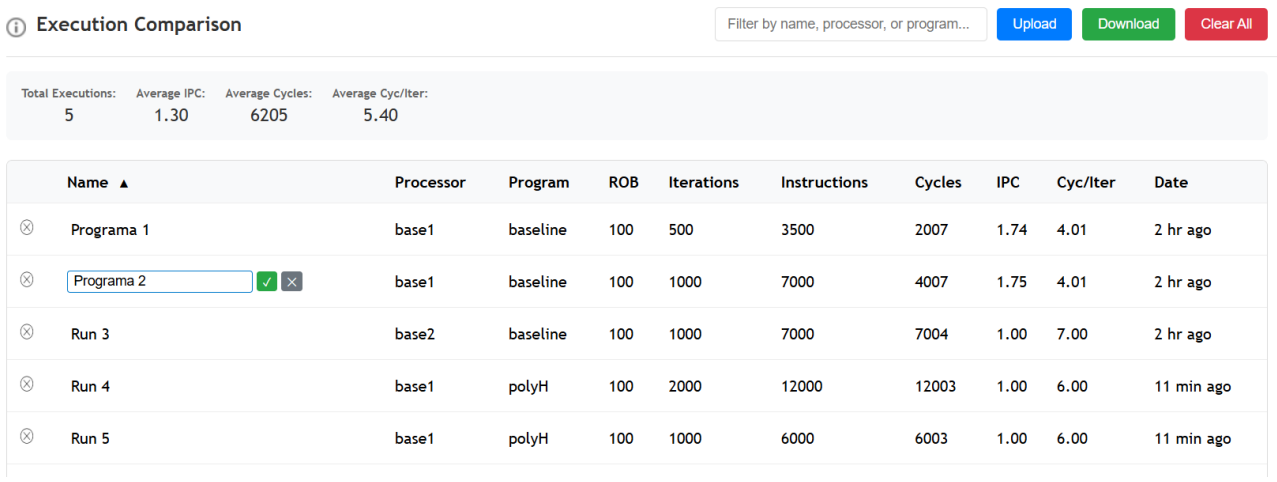

Fig. 7: Interfície del comparador de mètriques en disposició de dues columnes

- **Gràfics comparatius**: Afegir visualitzacions interactives per comparar mètriques de rendiment entre diferents execucions, facilitant la identificació de tendències i patrons.
- **Exportació d'informes**: Generar documents en format PDF amb les mètriques i configuracions de les simulacions, útil per a la impressió.

6 Conclusions

Aquest treball ha assolit els objectius principals. S'ha desplegat un sistema de tutorials interactius amb passos guiats i preguntes, i un editor visual que permet al professorat crear-los amb imatges i validacions sense tocar els fitxers JSON.

El sistema d'emmagatzematge local amb *LocalStorage* permet la persistència de dades entre sessions, unificant la gestió de programes i configuracions. La funcionalitat de comparació de mètriques facilita analitzar com afecten les diferents configuracions al rendiment.

L'editor de programes integrat a la interfície permet experimentar amb seqüències d'instruccions directament al navegador, tot i que la seva utilitat completa depèn de la futura implementació de la lògica de *cache* al mòdul RVCAT.

Pel que fa al simulador web, s'ha optat per mantenir l'aplicació completament *client-side*, sense servidor. Aquesta decisió simplifica el desplegament i el manteniment, ja que només cal allotjar fitxers estàtics (per exemple, a GitHub Pages) sense necessitat de gestionar infraestructura addicional.

En definitiva, RVCAT-WEB (disponible a `hpca4se-uab.github.io/rvcat-web-ui/`) és una eina educativa funcional i extensible per a l'aprenentatge d'arquitectura de processadors. La seva arquitectura modular facilita incorporar noves funcionalitats en el futur, seguint les línies de millora proposades.

Agraïments

Vull agrair als meus tutors, Juan Carlos Moure Lopez i Albert Jiménez Blanco, la seva orientació i suport durant el desenvolupament d'aquest projecte.

Referències

[1] GitHub Pages documentation. Consultat el setembre de 2025. https://docs.github.com/en/pages

[2] Vue.js Guide. Consultat el setembre de 2025. https://vuejs.org/guide/introduction.html

[3] JavaScript Tutorial – W3Schools. Consultat el setembre de 2025. https://www.w3schools.com/js/

[4] HTML Tutorial – W3Schools. Consultat el setembre de 2025. https://www.w3schools.com/html/

[5] CSS Tutorial – W3Schools. Consultat el setembre de 2025. https://www.w3schools.com/css/

[6] Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research*, 77(1), 81–112. https://doi.org/10.3102/003465430298487

[7] Tantau, T. (2023). The TikZ and PGF Packages - Manual for version 3.1.10. https://ftp.cvut.cz/tex-archive/graphics/pgf/base/doc/pgfmanual.pdf

[8] MDN Web Docs. Window: localStorage property. Consultat el gener de 2026. https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

Apèndix

A.1 Ús de la IA generativa en el desenvolupament de l'informe i del treball

S'ha utilitzat Claude (Anthropic) com a assistent de programació i per generar els diagrames TikZ d'aquest informe.

B.2 Interfície de l'editor de tutorials

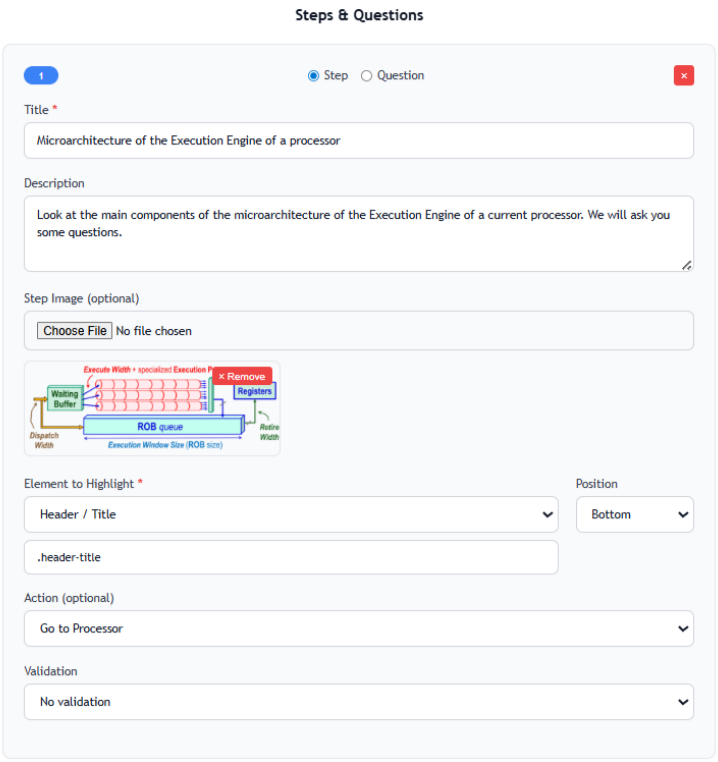

Fig. 8: Interfície de l'editor de tutorials: edició d'un pas (*step*)

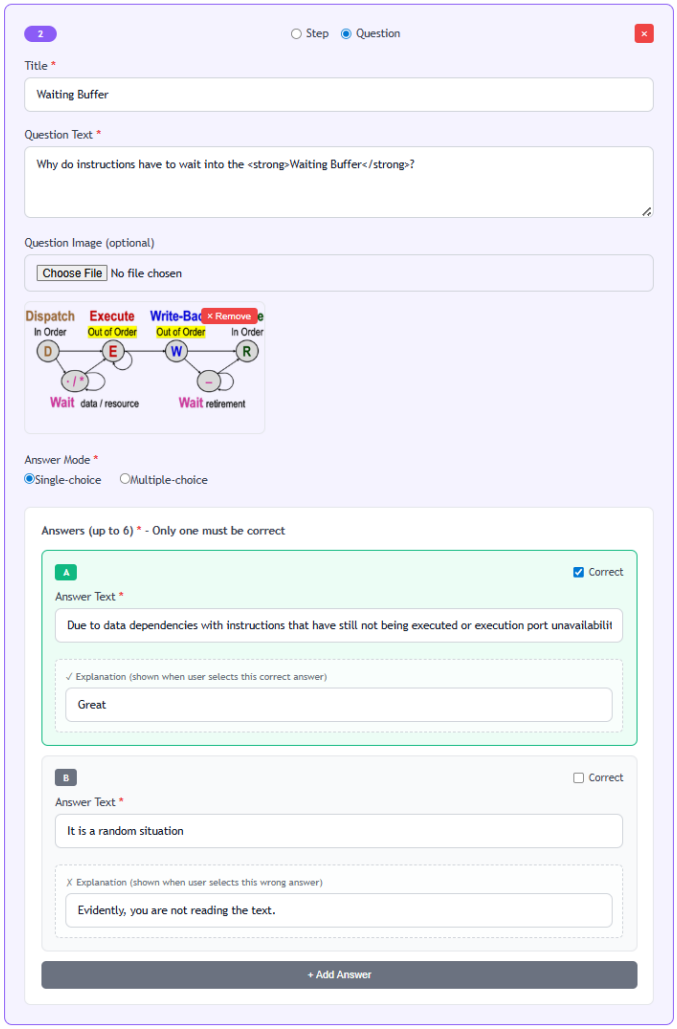

Fig. 9: Interfície de l'editor de tutorials: edició d'una pregunta (*question*)